DOCUMENT RESUME

ED 349 968 IR 015 847

AUTHOR Swan, Karen; Black, John B.
TITLE Logo Programming, Problem Solving, and Knowledge-Based Instruction.
PUB DATE Apr 90
NOTE 39p.; Paper presented at the Annual Meeting of the American Educational Research Association (Boston, MA, April 16-20, 1990).
PUB TYPE Reports - Research/Technical (143) -- Speeches/Conference Papers (150)

EDRS PRICE MF01/PC02 Plus Postage.
DESCRIPTORS *Abstract Reasoning; Computer Assisted Instruction; Educational Strategies; Expert Systems; Intermediate Grades; Junior High Schools; Microcomputers; *Problem Solving; Programing Languages; *Thinking Skills
IDENTIFIERS *LOGO Programing Language

ABSTRACT

The research reported in this paper was designed to investigate the hypothesis that computer programming may support the teaching and learning of problem solving, but that to do so, problem solving must be explicitly taught. Three studies involved students in several grades: 4th, 6th, 8th, 11th, and 12th. Findings collectively show that five particular problem solving strategies can be developed in students explicitly taught those strategies and given practice applying them to solve Logo programming problems. The research further demonstrates the superiority of such intervention over Logo programming practice alone, explicit strategy training with concrete manipulative practice, and the instruction in content areas that is traditionally prescribed for the teaching and learning of problem solving. Knowledge-based instruction linking declarative to procedural knowledge of problem solving strategies is recommended as a means to this end. The results also suggest, however, that computing environments may be uniquely conducive to the development of problem solving skills as they help learners bridge the gap between concrete and formal understanding. (Contains 45 references.) (DB)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

U.S. DEPARTMENT OF EDUCATION
Office of Educational Research and Improvement
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)

☐ This document has been reproduced as received from the person or organization originating it.

☐ Minor changes have been made to improve reproduction quality.

• Points of view or opinions stated in this document do not necessarily represent official OERI position or policy.

ED349968

LOGO PROGRAMMING, PROBLEM SOLVING, AND KNOWLEDGE-BASED INSTRUCTION

Karen Swan, University at Albany
John B. Black, Teachers College, Columbia University

ABSTRACT

Not very long ago, computer programming was touted as the solution to the problem solving crisis in American education, a discipline through which students would automatically acquire logical thinking and problem solving skills. More recently, however, such notions have gone the way of similar ideas concerning Latin and geometry. Research has indicated that problem solving abilities are not automatically acquired through computer programming, and programming is accordingly being de-emphasized in computer education. Some researchers, however, maintain that computer programming might well support the teaching and learning of problem solving, but that to do so, problem solving must be explicitly taught. The research reported in this paper was designed to investigate such hypothesis. Three studies are described which collectively show that five particular problem solving strategies can be developed in students explicitly taught those strategies and given practice applying them to solve Logo programming problems. The research further demonstrates the superiority of such intervention over Logo programming practice along, explicit strategy training with concrete manipulatives practice, and instruction in content areas traditionally prescribed for the teaching and learning of problem solving. The results indicate that problem solving strategies will not be developed through Logo programming alone, rather must be explicitly taught and practiced. Knowledge-based instruction linking declarative to procedural knowledge of problem solving strategies is recommended as a means to this end. The results also suggest, however, that computing environments may be uniquely conducive to the development of problem solving skills, in that they support quasi-concrete, malleable representations of abstract concepts that can help learners bridge the gap between concrete and formal understanding.

IR015847

"PERMISSION TO REPRODUCE THIS MATERIAL HAS BEEN GRANTED BY
Karen Swan

TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)."

BEST COPY AVAILABLE

BACKGROUND

In 1980, Seymour Papert published Mindstorms, a book which excited many educators with its notion that computer programming was fertile ground for the development of problem solving abilities. In Mindstorms, Papert (1980) made two claims. First, he argued that the computer was a revolutionary educational tool because it supported "transitional objects to think with," quasi-concrete representations of abstract ideas that could help learners bridge the gap between concrete and formal thought. Second, he argued that the transition from concrete to formal thinking would take place automatically, "painlessly, and without organized instruction" when learners were given computing environments rich in such transitional objects to explore, environments such as Papert's own programming language, Logo.

Papert's claims have engendered much debate both within and outside of the Logo community. The research reported in this paper was designed to investigate each of these claims in relation to the teaching and learning of particular problem solving strategies. We believe the importance of separating the claims is born out by our results which indicate that the first claim -- that computing environments can uniquely support the development of problem solving skills -- is true, whereas the second claim -- that this development will happen automatically -- is not. These results make sense of the seemingly contradictory reports found in the literature concerning Logo programming and the teaching and learning of problem solving; that investigations which looked for the automatic development of problem solving skills resulting from Logo programming experience found no such occurrence (Papert, Watt, diSessa & Weir, 1979; Pea & Kurland, 1984; Leron, 1985), but that those which investigated mindful interventions combining explicit instruction with Logo programming practice reported positive results (Carver & Klahr, 1986; Thompson & Wang, 1988; Black, Swan & Schwartz, 1988; de Corte, Verschaffel & Schrooten, 1989; Lehrer, Sancillio & Randle, 1989).

In this paper, we will examine the theoretical foundations for our findings concerning each of the two claims. We will then discuss the particular problem solving strategies we investigated and the problem solving intervention we designed to teach them. Finally, we will report on three studies we conducted to test Papert's (1980) claims, and on our conclusions based on their findings.

"Transitional Objects to Think With."

A quarter of a century ago, Marshall McLuhan (1964) advanced the notion that communications media are extensions of the human sensory apparatus which alter the ways in which we use our senses, and so, the manner in which we perceive ourselves and our environment. Language, for example, alters the way we perceive the world by naming and defining objects in it, thus allowing us to distinguish between them in ways that would not be possible without language. McLuhan argued that the formal nature of a particular medium of

communication was therefore more influential in shaping our thought than the content of the message it transmitted.

Whether or not one accepts McLuhan's hypothesis en toto, it seems clear that each medium of communication entails unique formal attributes that matter, or that can be made to matter, in learning. Salomon (1981), for example, has shown how differing filmic presentations can "activate," "short-circuit," or "model" particular cognitive processes. In this vein, the computing medium has been singled out in recent years as particularly supportive of the development of problem solving abilities (Feurzig, Horowitz & Nickerson, 1981; Harvey, 1982; Mayer, Dyck & Vilberg, 1986; Linn, 1988; Soloway, 1986). The Logo programming language, in particular, has been described as an environment designed to "help children to develop problem solving skills, to think more clearly, to develop an awareness of themselves as thinkers and learners" (Watt, 1983, p. 48).

Indeed, Seymour Papert (1980), Logo's creator, maintains that computers are truly revolutionary educational tools because they support "transitional objects to think with." His idea seems to be that abstract ideas can be represented, manipulated, and dynamically tested in computing environments, thereby relieving burdens to working memory and providing students with quasi-concrete models of cognitive processes that can be easily internalized. Subgoals formation, for example, is a problem solving strategy that is given quasi-concrete representation in Logo programming in that Logo programs are composed of small subprocedures each of which define the means to satisfying particular parts of a larger programming problem. Each subprocedure, moreover, can be written, tested, and refined by itself before the parts are assembled to create the larger program. Programming in this manner, it can be argued, provides a model of the abstract process of subgoals formation which can be easily internalized and generalized. Papert writes (1980, p. 23), "I began to see how children who had learned to program computers could use very concrete computer models to think about thinking and to learn about learning . . ."

Papert moreover contends (1980, p. 9), ". . . that the computer presence will enable us to so modify the learning environment outside the classroom that much if not all the knowledge schools presently try to teach with such pain and expense and such limited success will be learned, as the child learns to talk, painlessly, successfully, and without organized instruction." Unfortunately, researchers investigating the effect of computer programming on children's problem solving skills found no such link between programming practice and the automatic development of problem solving abilities (Papert, Watt et al, 1979; Ehrlich, Abbott, Salter, and Soloway, 1984; Pea & Kurland, 1984; Leron, 1985; Patterson & Smith, 1986; Shaw, 1986; Mandinach & Linn, 1987). Their findings have led many educators to discount all notions of the unique suitability of programming environments for the development of thinking and problem solving skills, and programming is being accordingly de-emphasized in educational computing

BEST COPY AVAILABLE

programs. In the opinion of certain authors, however, this is a case of throwing out the baby with the bath water. Programming environments, they argue, most probably are uniquely suited to the development of certain higher-order cognitive processes, but such processes are not developed automatically, rather must be mindfully taught and practiced (Pea & Kurland, 1984; Leron, 1985; Salomon & Perkins, 1987).

Indeed, investigations including this sort of mindful intervention have demonstrated that Logo programming can support the development of rule-learning (Gorman & Bourne, 1983; Degelman, Free, Scarlato, Blackburn & Golden, 1986), reflectivity (Clements & Gullo, 1984; Miller & Emihovich, 1986), divergent thinking (Clements & Gullo, 1984; Swan & Black, 1989), problem representation (de Corte et al, 1989), analogical reasoning (Clements, 1987; Swan & Black, 1989), debugging skills (Carver, 1987; Swan & Black, 1989; de Corte et al, 1989), subgoals formation (Swan & Black, 1989; de Corte et al, 1989), and forward chaining strategies (Swan & Black, 1989); as well as the capacity to apply learned concepts to novel tasks (Lehrer & Randle, 1987; Thompson & Wang, 1988; Lehrer et al, 1989). Moreover, they have shown that problem solving instruction with Logo programming practice better supports such development than conventional instruction and practice. These findings suggest that computing environments in general and the Logo programming environment in particular can be uniquely supportive of the teaching and learning of problem solving skills as claimed, but only when those skills are specifically identified and mindfully taught and practiced.

"Knowledge-Based Instruction."

Such findings are really nothing new; they mirror Thorndike's (1907) results concerning Latin and logical thinking. Thorndike concluded that the knowledge of higher order skills such as logical thinking or problem solving would only be internalized and transferred to other domains to the extent to which that knowledge was explicitly taught. To explicitly teach higher order skills, one needs to specify the knowledge those processes require; one needs to premise the design of instruction on knowledge outcomes, rather than on behavioral outcomes.

We believe the distinction is a real one. Problem solving instruction, for example, has traditionally sought behavioral outcomes. It has focused on the increased ability to solve particular kinds of problems and has accordingly concentrated on practice solving those problems. This doesn't work. While the students in Thorndike's studies might have become better at solving particular Latin problems, they did not acquire generalized logical thinking skills that could be applied to problems in differing domains. Similarly, students in the early Logo studies (Papert et al, 1979; Pea & Kurland, 1984; Leron, 1985) might have become better at solving particular Logo programming problems, but those skills were not generalized and transferred to other domains.

It is our contention that they are not successful because complex cognitive behaviors like problem solving involve more than their manifest behaviors and must be addressed at a deeper

level, at the level of the knowledge structures which support such behaviors. This knowledge has two components -- a declarative component, the knowledge of what those specific steps entail, and a procedural component, the knowledge of how to perform them. What is missing from behaviorally-based problem solving instruction is a declarative focus, an explicit description of the steps involved in particular problem solving strategies. Indeed, Anderson (1983) argues that all knowledge begins as declarative knowledge and is gradually proceduralized through practice.

We believe that what is at issue here is the abstraction and generalization of knowledge; that an explicit declarative focus is necessary for building knowledge structures that are generalizable. We believe that it is not enough to merely give students procedural experience solving particular problems, but rather that such experience must be linked with the declarative knowledge of specific problem solving strategies, before those strategies can be generalized and applied in differing contexts. Just as the naming and defining of objects through language leads to the abstraction of their qualities so that they can be manipulated as objects of thought, so the naming and defining of specific problem solving processes decontextualizes them so that they can be generalized and applied in other domains. This is what we did in designing the Logo-based problem solving intervention we used in our studies. We began with a focus on the declarative knowledge underlying six specific problem solving strategies.

Problem Solving Strategies.

A number of distinct problem solving strategies can be distinguished within behaviors considered general problem solving (Wicklegren, 1974; Newell & Simon, 1972). Certain of these seem more applicable to programming problems in general, children's programming in particular (Clements & Gullo, 1984; Lawler, 1985; Clement, Kurland, Mawby & Pea, 1986). We performed task analyses based on Polya's (1973) decomposition of the problem solving process of the six particular problem solving strategies we believed would be most applicable to Logo programming. These were subgoals formation, forward chaining, backward chaining, systematic trial and error, alternative representation, and analogy. A description of each follows.

Subgoals formation.

Subgoals formation refers to breaking a single difficult problem into two or more simpler problems. Subgoals formation might thus be seen as the mapping of a problem space. Even when no obviously solvable subgoals can be found, breaking a problem into its constituent parts makes its solution less formidable, more manageable, and less susceptible to errors. Subgoals formation can be described by the following four steps:

1. Problem definition. Specify the problem.

2. Subdivision. Examine the problem specification to see where it can be broken into smaller, self-contained problems. Specify these and their connections to the larger problem.

3. Evaluation. Test the subproblems generated for grain size and further decomposition. If the subproblems are manageable or cannot be further decomposed, solve them. Recombine these partial solutions into the total solution using the connections specified in step 2.

4. Recursion. Otherwise, repeat the second and third steps for each of the subproblems generated. Continue the process until no more smaller problems can be generated for any of the subproblems.

While subgoals formation might seem an obvious strategy to adults, it is not at all obvious to many children (Carver & Klahr, 1986), therefore it is a good candidate for explicit instruction. Moreover, of all the problem solving strategies, it can most clearly be implemented and concretized in Logo programming. In Logo, small subprocedures are easily written and placed in the workspace. Because these can be called from anywhere in a program, a program can simply be a list of such subprocedures, a very concrete representation of the subgoals that make up a programming solution.

Forward chaining.

Forward chaining is a species of means-ends analysis (Newell & Simon, 1972) which involves working from what is given in a problem towards the problem goal in step-by-step, transformational increments that bring one progressively closer to that goal. The forward chaining process can be decomposed into the following steps:

1. Problem definition. Specify the problem goal. Specify what is given. Specify the constraints, if any.

2. Transformation. Use domain operators to manipulate the givens to bring them closer to the goal state.

3. Evaluation. Compare the desired goal, the givens, and the transformation. Test to see whether the transformation is really closer to the goal than the givens. If it is not, redo step 2.

4. Recursion. Make the transformation a new given. Repeat steps 2 and 3 using the new given. Continue in this manner until the goal state is reached and the problem is solved.

A programming environment, especially an interpreted environment like Logo, is inherently supportive of the forward chaining process. Transformations can be implemented, their effects accessed, and successful changes instantiated as partial programs, with relative ease and little risk. A program can thus be developed in incremental steps and such development provide a concrete model of the forward chaining process. An important part of forward chaining, however, involves the ability to choose appropriate transformations and evaluate whether or not these actually bring one nearer problem solution. Forward chaining thus requires some sort of mental model of the problem space, and is not, therefore, typically a novice technique (Greeno & Simon, 1984).

Backward chaining.

Backward chaining, however, is typically a novice problem solving strategy (Greeno & Simon, 1984). Backward chaining focuses on the goal state and tries to deduce a preceding state

from which that goal could be derived, then a state from which that state could be derived, and so on, working backward to what is given in a problem. The backward chaining process consists of the following four steps:

1. Problem definition. Specify the goal state. Specify what is given in the problem.

2. Decomposition. Specify a state that could be transformed into the goal state in a single step.

3. Evaluation. Test the specified transformation to be sure it can in fact be transformed into the goal state. If it cannot, redo step 2. Examine the specified transformation to make sure it is closer to what is given than is the goal. If it isn't, redo step 2.

4. Recursion. Otherwise, make the specified state a new goal state and repeat steps 2 and 3 using this new goal state. Continue in this manner until a clear path from givens to goal can be discerned. Use it to solve the problem.

Backward chaining is a particularly useful technique when a problem has a uniquely specified goal, and/or is a situation in which the inputs and outputs of the transformations can be uniquely specified (Wicklegren, 1974). Such is often the case in programming problems. Indeed, programming has been identified as a teleological domain (Bolter, 1984). Moreover, in as much as it is typically a novice technique, backward chaining might be more available to children. However, backward chaining has no real analog in Logo programming as one cannot build programs backwards.

Systematic trial and error.

Systematic trial and error involves the recursive testing of possible solutions in a systematic, guided fashion, and the problem reduction and/or refinement resulting from such tests. It differs from forward chaining in that it includes negative inference (problem reduction through the elimination of possibilities) as well as positive inference, and in that it's power lies in the systematic testing of such possibilities rather than in the choice of the best of these. The steps involved in the systematic trial and error process include:

1. Problem definition. Specify the problem goal.

2. Approximate solution. Create and implement a plan to solve the problem.

3. Evaluation. Compare the problem goal with the instantiated solution. If there are no discrepancies between them, the problem is solved. Otherwise, generate a description of the discrepancies between the desired goal and the instantiated solution.

4. Recursion. Use the description of goal/solution discrepancies to revise the plan, and reapply steps 2 and 3. Continue in this manner until the instantiated solution matches the desired goal.

Piaget (Ginsberg & Opper, 1979) believed that the application of systematic trial and error strategies was an important determinant of formal operational ability. Systematic trial and error, then, is an obvious candidate for testing Papert's (1980) notion that programming environments

support the concretizing of the formal. Certain types of graphics programming, moreover, are paradigmatic of systematic trial and error strategies. Debugging also makes use of, and provides symbolic representations for, such techniques (Carver, 1987).

Alternativerepresentation.

Alternative representation involves conceptualizing a problem from differing perspectives. Polya (1973) writes that often the way a problem is stated is really all that makes it difficult, that simple restatement will make its solution obvious. Alternative representation is thus the antidote to functional fixedness (Dunker, 1945). Alternative representation can be decomposed into the following four-step description:

1. Problem definition. Specify the problem.

2. Alternative representation. Generate an alternative problem specification

3. Evaluation. Test to see whether the new problem specification suggests problem solution. If it does, solve the problem.

4. Recursion. Otherwise, repeat steps 2 and 3 by generating and evaluating other problem specifications until a problem solution is found.

Programming is conducive to the development of alternative representations both because there are never single correct solutions to programming problems, and because differing representations can quite easily be instantiated and pragmatically tested in programming environments. Indeed, Clements and Gullo's (1984) study of the effects of Logo programming on young children's cognition found significant increases in their ability to produce alternative representations. Statz's (1973) finding of significant increases on permutation tasks may also support this view. However, their is no clear analog in Logo programming for alternative representationstrategies.

Analogy.

Analogy involves the discovery of a particular similarity between two things otherwise more or less unlike, and "a mapping of knowledge from one domain (the base) onto another (the target) predicated on a system of relations that holds among the objects of both domains." (Gentner, 1987) An important factor in this process, especially in problem solving contexts, is goal-relatedness, how one domain is like another with respect to a specified goal (Holyoak & Koh, 1987). The use of analogy in problem solving can be decomposed into the following steps:

1. Problem definition. Specify the desired goal. Specify the base and the target systems.

2. Mapping. Perform a mapping between the base and target systems.

3. Evaluation. Test the soundness of the match in terms of both structural similarity and pragmatics (goal related conditions). If the analogy generated meets the goal conditions, and the structural similarity between the base and the target holds, the mapping is sound. Use the base domain solution to generate a solution in the target domain.

4. Recursion. Otherwise, return to step 1 and specify a new base domain. Apply steps 2 and 3 to it. Continue in this manner until an adequate representation is discovered.

Programming environments inherently support the development of analogy in that one is always mapping between computer code (a formal representation) and program output (a concrete representation), and recursion, an important Logo programming technique, is a form of analogical reasoning. Recursion, however, is confusing to most adults. None-the-less, Doug Clements (1987) found significantly better analogical reasoning among students with prior Logo experience. Analogy, moreover, is the basis for all transfer, thus a critical element in the research we designed.

***** INSERT FIGURE 1 ABOUT HERE *****

These six problem solving strategies -- subgoals formation, forward chaining, backward chaining systematic trial and error, alternative representation, and analogy can be more or less concretely represented, then, within a Logo programming context. We accordingly designed our instruction and our testing procedures around them. The instruction was split into units, one for each strategy. Each unit began with explicit instruction on a particular strategy (declarative knowledge) in which wall posters which enumerated the steps involved in each strategy (FIGURE 1) were presented and discussed. This was followed by mediated practice (Feuerstein, 1980; Delclos, Littlefield & Bransford, 1985; Corno, 1986) solving problems designed to be particularly amenable to solutions employing that strategy (procedural knowledge). An example of a problem set is given in FIGURE 2. We likewise created six separate tests, each designed to measure students' facility in applying specific strategies to non-computing problems. (See APPENDIX.) Our goal was for students to transfer the strategies learned in the intervention to the paper and pencil tasks of the problem solving tests.

***** INSERT FIGURE 2 ABOUT HERE *****

STUDY ONE

Study One was a pilot study concerned with testing the efficacy of the intervention we designed for supporting the development of six particular problem solving skills. Within that context, two factors we believed might effect such development were also manipulated and examined. The first of these was the specific Logo practice environments utilized. In particular, we thought that students might be more likely to develop problem solving strategies practiced within both the graphics and the list-processing programming environments, but we also thought it was possible that one or the other of these environments alone would be more supportive of the development of particular strategies. The second factor we thought might influence the effectiveness of the instruction was the grade levels of the subjects involved. Specifically, we thought there might be developmental differences in students' propensity to develop particular problem solving strategies. Study One thus examined three interrelated questions:

1. Can the Logo programming environment support the development of subgoals formation, forward chaining, backward chaining, systematic trial and error, alternative representation, and/or analogy strategies when those strategies are explicitly taught and practiced?

2. Do differing practice environments within Logo programming differentially effect the likelihood that such strategies will be developed within that instructional context?

3. Do developmental differences differentially effect the the development of such strategies will be developed within that instructional context?

Subjects.

Subjects were 133 fourth through eighth grade students in a private elementary school. All subjects had at least 30 hours previous Logo programming experience.

Methodology.

All subjects were given paper-and-pencil exercises testing their ability to apply six problem solving strategies -- subgoals formation, forward chaining, backward chaining, systematic trial and error, alternative representation, and analogy -- and randomly assigned by grade to one of three treatment groups receiving respectively graphics, list processing, or graphics and list processing practice problems. All subjects received training in each strategy, then were asked to solve four practice problems particularly amenable to solutions involving that strategy. On completion of all six strategy units, subjects were post-tested using different but analogous exercises. Mean pre-test scores were compared between groups using one-way analysis of variance and found to be statistically equivalent ($F_{2, 130} = 1.50$, $p > .10$), hence, treatment groups were assumed to be generally equal in problem solving ability before treatment. Raw scores on all tests except those for alternative representation were converted to percent correct scores and compared using a four-way analysis of variance. Independent variables were test, strategy, class (grade level), and group. The dependent variables were scores on the strategies tests. Because they had no maximum possible correct, alternative representations measures were evaluated separately using a three-way analysis of variance.

Results.

Significant differences were found between subjects' mean pre- and post-test scores for all problem solving strategies except backward chaining for subjects in all treatment groups. These results demonstrate the effectiveness of the instruction we designed for supporting the development of subgoals formation, forward chaining, systematic trial and error, alternative representation, and analogy strategies. The intervention was not shown to be effective for the teaching and learning of backward chaining strategies, nor were differing practice environments found to differentially effect strategy development. The results also revealed developmental differences in students' facilities for both using and developing particular problem solving strategies. Not surprisingly, older students were better than

younger ones at applying all strategies. They were also more likely to benefit from instruction and practice in alternative representation and analogy strategies, while younger students benefited more than older ones from instruction and practice in subgoals formation strategies.

TABLE 1
Study One; ANOVA Table for Subgoals Formation, Forward Chaining, Backward Chaining, Systematic Trial & Error, and Analogy

	SS	DF	MS	F	PROB
MEAN	4952023.5	1	4952023.5	3784.56	0.0000
CLASS	67283.5	4	16820.9	12.86	0.0000
GROUP	4388.2	2	2191.1	1.68	0.1914
CG	18297.7	8	2287.2	1.75	0.0944
ERROR	154400.6	118	1308.2		
TEST	40134.8	1	40134.8	170.86	0.0000
TC	947.3	4	236.8	1.01	0.4061
TG	490.9	2	245.5	1.04	0.3549
TCG	1286.8	8	160.8	0.68	0.7042
ERROR	27717.9	118	234.9		
STRATEGY	229966.2	4	57491.6	171.72	0.0000
SC	41420.0	16	2588.7	7.73	0.0000
SG	2205.1	8	275.6	0.82	0.5823
SCG	15784.3	32	492.1	1.47	0.0495
ERROR	158028.5	472	334.8		
TS	14457.3	4	3614.3	14.79	0.0000
TSC	10124.9	16	632.8	2.59	0.0007
TSG	2064.6	8	258.1	1.06	0.3928
TSCG	7842.1	32	245.1	1.01	0.4654
ERROR	115325.9	472	244.3		

TABLE 1 shows the results of the four-way analysis of variance comparing students' strategy measure scores in Study One. Significant main effects for class ($F_{2,118} = 12.86$, $p < .01$), test ($F_{1,118} = 170.86$, $p < .01$), and strategy ($F_{4,472} = 1.71.72$, $p < .01$) were found. No significant main group effect was discovered ($F_{2,118} = 1.68$, $p > .10$). TABLE 2 shows the results of the three-way analysis of variance comparing students' alternative representation scores.. Again significant main effects were found for class ($F_{4,118} = 6.02$, $p < .01$) and test ($F_{1,118} = 1.37$, $p < .01$), but no main group effect was found ($F_{2,118} = 1.05$, $p > .10$).

The main class effect indicates developmental differences in students' problem solving abilities. Students' mean scores for all problem solving strategies rose with increasing grade levels, indicating that older students were more adept at utilizing the strategies than younger ones. This effect, although expected, supports views linking problem solving with formal operational skills. The main test effect indicates significant pre- to post-test differences across strategies for all grade levels and treatment groups. Such increases argue for the success of the instruction tested. The main strategy effect indicates significant differences between mean scores on the various

strategy measures. Because these measures were not designed to be equivilant, however, the effect is not meaningful. The lack of a group effect indicates no significant differences between treatment groups, thus that differing practice environments within Logo had no effect on the efficacy of the instruction we designed.

TABLE 2
Study One; ANOVA Table for Alternative Representation

	SS	DF	MS	F	PROB
MEAN	3389956.9	1	3389956.9	2248.39	0.0000
CLASS	36310.0	4	9077.5	6.02	0.0002
GROUP	3170.5	2	21589.2	1.05	0.3518
CG	19551.9	8	2444.0	1.62	0.1259
ERROR	177911.7	118	1507.7		
TEST	115234.5	1	115234.5	137.78	0.0000
TC	14196.0	4	3549.0	4.24	0.0030
TG	1599.8	2	799.9	0.96	0.3873
TCG	7734.4	8	966.8	1.16	0.3316
ERROR	98694.5	118	836.4		

The four-way analysis of variance also examined eleven interaction effects. Significant strategy by class ($F_{16,472} = 7.73$, $p < .01$), test by strategy ($F_{14,472} = 14.79$, $p < .01$), and test by strategy by class ($F_{16,472} = 2.59$, $p < .01$) effects were discovered. The test by strategy effect indicates differences in pre- to post-test changes between the five problem solving strategies tested. An examination of the simple test effect at each level of strategy reveals significant test effects for all of these except backward chaining, indicating that students developed subgoals formation, forward chaining, systematic trial and error, and analogy strategies as a result of the intervention (FIGURE 3). A similar result was found for measures of alternative representation. Such results argue strongly for the success of the intervention, and support the finding of a main test effect.

***** INSERT FIGURE 3 ABOUT HERE *****

Likewise, a closer examination of the strategy by class interaction supports the finding of a main effect for class. Simple class effects were found at all strategy levels except backward chaining, indicating developmental differences in students' problems solving abilities, with older students generally scoring higher than younger ones on the various strategy measures (FIGURE 4). In addition, a finer grained analysis of the test by strategy by class interaction indicates developmental differences in students' propensity to develop particular strategies. Progressively weaker test effects were found at increasingly higher grade levels for subgoals formation measures, and progressively stronger test effects were found at increasingly higher grade levels for measures of analogy. A seperate analysis also revealed progressively stronger test effects at increasingly higher grade levels for alternative representation measures. Such results hint at developmental differences in students' readiness to develop certain problem solving strategies, in particular, that younger studentswere more ready to develop subgoals formation strategies, and that

older students were more ready to develop alternative representation and analogy strategies.

***** INSERT FIGURE 4 ABOUT HERE *****

Discussion

The results of Study One strongly suggest that the instruction we designed supports the development of five particular problem solving strategies -- subgoals formation, forward chaining, systematic trial and error, alternative representation, and analogy -- but not the development of backward chaining strategies, among the population tested. While the positive increases observed on these strategy measures might also have resulted from practice and/or maturation, the size of the increases and the lack of similar increases on backward chaining measures argue that such was not the case. The results also suggest that differing practice environments within Logo programming have little effect on the success of such intruction. Finally, the results point to developmental differences in students' facility with and propensity to develop particular problem solving skills. Specifically, we found that older students were more likely to develop alternative representation and analogy strategies, whereas younger students were more likely to develop subgoals formation strategies.

STUDY TWO

Study Two was concerned with validating the results of Study One with reference to meaningful controls; in particular, with assessing the importance of Logo programming within the instruction we designed, and with differentiating between that intervention and discovery learning approaches. Study Two investigated the differing efficacies of explicit problem solving instruction with Logo programming practice vs. discovery learning in a Logo programming environment, and the Logo programming environment vs. a concrete manipulatives practice environment for supporting the development of the five problem solving strategies on which students showed improvement in Study One -- subgoals formation, forward chaining, systematic trial and error, alternative representation, and analogy. Two research questions were addressed:

1. Is explicit problem solving instruction with Logo programming practice superior to discovery learning in a Logo programming environment for supporting the development of five particular problem solving strategies -- subgoals formation, forward chaining, systematic trial and error, alternative representation, and analogy?

2. Is the Logo programming domain particularly supportive of the teaching and learning of problem solving?

Subjects.

Subjects were 100 fourth through sixth grade students at the same private elementary school where Study One was completed. All had at least 30 hours previous Logo programming experience.

older students were more ready to develop alternative representation and analogy strategies.

***** INSERT FIGURE 4 ABOUT HERE *****

Discussion

The results of Study One strongly suggest that the instruction we designed supports the development of five particular problem solving strategies -- subgoals formation, forward chaining, systematic trial and error, alternative representation, and analogy -- but not the development of backward chaining strategies, among the population tested. While the positive increases observed on these strategy measures might also have resulted from practice and/or maturation, the size of the increases and the lack of similar increases on backward chaining measures argue that such was not the case. The results also suggest that differing practice environments within Logo programming have little effect on the success of such intruction. Finally, the results point to developmental differences in students' facility with and propensity to develop particular problem solving skills. Specifically, we found that older students were more likely to develop alternative representation and analogy strategies, whereas younger students were more likely to develop subgoals formation strategies.

STUDY TWO

Study Two was concerned with validating the results of Study One with reference to meaningful controls; in particular, with assessing the importance of Logo programming within the instruction we designed, and with differentiating between that intervention and discovery learning approaches. Study Two investigated the differing efficacies of explicit problem solving instruction with Logo programming practice vs. discovery learning in a Logo programming environment, and the Logo programming environment vs. a concrete manipulatives practice environment for supporting the development of the five problem solving strategies on which students showed improvement in Study One -- subgoals formation, forward chaining, systematic trial and error, alternative representation, and analogy. Two research questions were addressed:

1. Is explicit problem solving instruction with Logo programming practice superior to discovery learning in a Logo programming environment for supporting the development of five particular problem solving strategies -- subgoals formation, forward chaining, systematic trial and error, alternative representation, and analogy?

2. Is the Logo programming domain particularly supportive of the teaching and learning of problem solving?

Subjects.

Subjects were 100 fourth through sixth grade students at the same private elementary school where Study One was completed. All had at least 30 hours previous Logo programming experience.

BEST COPY AVAILABLE

Methodology.

All subjects were tested on their ability to apply the five problem solving strategies on which subjects improved in the first study -- subgoals formation, forward chaining, systematic trial and error, alternative representation, and analogy. They were then randomly assigned by grade to one of three treatment groups receiving respectively explicit problem solving instruction with Logo graphics practice, explicit problem solving instruction with cut-paper manipulatives practice, or Logo graphics programming problems without strategy training. On completion of all treatments, subjects were post-tested using different but analogous tests. Mean pre-test scores on measures of subgoals formation, forward chaining, systematic trial and error, and analogy were compared between groups using one-way analysis of variance and found to be statistically equivalent ($F_{2, 97} = 0.33$, $p > .10$), hence, treatment groups were assumed to be generally equal in these problem solving abilities before treatment. Subjects' scores on the alternative representation pre-test, however, were not statistically equivalent ($F_{2,97} = 4.99$, $p < .01$). Students receiving problem solving instruction and Logo programming practice scored lower on the pre-test than students receiving Logo programming practice alone who scored lower than students receiving problem solving instruction and cut-paper manipulatives practice.

On completion of all treatments, subjects were post-tested using different but analogous tests. Raw scores on all tests except those for alternative representation were converted to percent correct scores and compared using a three-way analysis of variance. The independent variables were test, strategy, and group. The dependent variables were scores on the strategies tests. Because they had no maximum possible correct, alternative representations measures were evaluated seperately using a two-way analysis of variance.

Results.

Significant differences in pre- to post-test increases were found between treatment groups indicating that subjects receiving explicit problem solving instruction and Logo programming practice, and that group alone, improved in subgoals formation, forward chaining, systematic trial and error, and analogy strategies. Increased ability in applying alternative representation strategies was also indicated for this group but not conclusively demonstrated. The results thus argue for the superiority of explicit instruction and Logo programming practice over both similar instruction with cut-paper manipulatives practice and discovery learning in a similar practice environment for the teaching and learning of problem solving.

TABLE 3 shows the results of the three-way analysis of variance comparing students' strategy measure scores in Study Two. Significant main effects were found for all independent variables (group, $F_{2,97} = 12.81$, $p < .01$; test, $F_{1,97} = 5.94$, $p < .05$; strategy, $F_{3,291} = 207.11$, $p < .01$), indicating significant differences along all three dimensions. Of these, only the group effect is particularly meaningful. The strategy effect is not meaningful because the differing

strategy measures were not designed to be equivilant. The test effect indicates significant pre- to post-test changes, but these could have resulted from practice or maturity and not from the various interventions.

TABLE 3
Study Two; ANOVA Table for Subgoals Formation, Forward Chaining, Backward Chaining, Systematic Trial & Error, and Analogy

	SS	DF	MS	F	PROB
MEAN	2074999.0	1	2074999.0	1568.08	0.0000
GROUP	33907.5	2	16953.7	12.81	0.0000
ERROR	128357.9	97	1323.3		
TEST	1576.6	1	1576.6	5.94	0.0166
TG	6871.7	2	3437.3	12.96	0.0000
ERROR	25735.0	97	265.2		
STRATEGY	283190.9	3	94397.0	207.11	0.0000
SG	13478.6	6	2246.4	4.93	0.0001
ERROR	132630.9	291	455.8		
TS	2274.2	3	758.1	3.64	0.0132
TSG	743.4	6	123.9	0.60	0.7338
ERROR	60540.8	291	208.0		

The group effect, however, clearly indicates differences between groups resulting from the various interventions. Because the groups were statistically equivalent before receiving treatment, but significantly different overall, the group effect indicates differences in problem solving strategy skills resulting from differences in the interventions. This result is corroborated by the finding of a test by group interaction ($F_{2,97} = 12.96$, $p < .01$), indicating differences in pre- to post-test changes resulting from the differing treatments. This interaction was examined in greater detail by assessing the simple test effect at each level of group. A strong test effect was found for the group recieving problem solving instruction and Logo programming practice, and for that group alone. Students in receiving problem solving instruction and Logo programming practice improved an average of 11.1 percentage points on the four strategy measures, while the scores of students receiving similar instruction but cut-paper manipulatives practice remained essentially the same, and the scores of students receiving Logo programming practice alone actually declined slightly, although not significantly. FIGURE 5 illustrates these results. The findings demonstrate that the intervention we designed, and that intervention alone, resulted in significant increases in students' problem solving abilities in four areas -- subgoals formation, forward chaining, systematic trial and error, and analogy. They argue for the superiority of explicit problem solving instruction and Logo programming practice over both similar instruction with manipulatives practice and Logo programming alone.

***** INSERT FIGURE 5 ABOUT HERE *****

The results of the analysis of variance for alternative representation measures (TABLE 4) is problematic. Significant main effects were found for group ($F_{2,97} = 4.13$, $p < .05$) and test ($F_{2,97} = 10.55$, $p < .01$). Because the groups were not equivalent to begin with, the group effect is not meaningful. Indeed, an examination of group means shows that all groups improved on alternative representation measures, thus, the test effect is not meaningful either. What would be meaningful would be a solid test by group interaction. Unfortunately, the analysis of variance reveals only weak significance for the interaction ($F_{2,97} = 2.57$, $.05 < p < .10$).

TABLE 4
Study Two; ANOVA Table for Alternative Representation

	SS	DF	MS	F	PROB
MEAN	19113259.6	1	19113259.6	563.35	0.0000
GROUP	28058.5	2	14029.3	4.13	0.0000
ERROR	329431.8	97	3396.2		
TEST	24819.1	1	24819.1	10.55	0.0016
TG	6631.9	2	3315.9	2.57	0.0821
ERROR	125363.8	97	12922.4		

The simple test effect was none-the-less assessed at each level of group to examine the test by group interaction in greater detail. A strong test effect was found for the group receiving problem solving instruction and Logo programming practice ($F_{1,97} = 18.91$, $p < .01$), whereas only a weak test effect was found for the group receiving problem solving instruction and cut-paper manipulatives practice ($F_{1,97} = 3.61$, $.05 < p < .10$), and no test effect at all was found for the group receiving Logo programming practice alone ($F_{1,97} = 1.81$, $p > .10$). FIGURE 5 illustrates the differences in pre- to post-test changes between groups. It can be seen that the group receiving problem solving instruction and Logo programming practice improved more than twice as much as either the group receiving instruction with cut-paper manipulatives practice or the group receiving Logo programming practice alone. Because students in the explicit instruction - Logo programming group had lower scores on alternative representation measures to begin with, however, the greater gains they made might be attributed to differential ability levels rather than the intervention. Thus, the most we can conclude is that it is possible that students in the Logo graphics group showed greater increases on measures of alternative representation as a result of the intervention we designed.

Discussion

The results of Study Two argue strongly for the superiority of explicit problem solving instruction and Logo programming practice over both similar instruction with manipulatives practice and discovery learning in Logo programming environments for the development of four problem solving strategies -- subgoals formation, forward chaining, systematic trial and error, and analogy. Indications are that such pedagogy may be most effective for the teaching and learning of

alternative representation strategies as well. In terms of the research questions, then, we can conclude that explicit problem solving instruction with Logo programming practice is more supportive than Logo discovery learning environments of the development of those strategies, and that the Logo programming environment itself is more supportive of such development than concrete manipulatives. The findings thus support Papert's (1980) contention that abstract ideas can be concretely represented on computers in ways that help students bridge the gap between concrete and formal thought, but argue against his claim that such transition will take place automatically when students work within Logo programming environments.

STUDY THREE

Because Studies One and Two utilized the same teachers and similar student populations, we had some question about the general application of the instruction we designed. Study Three was thus concerned with validating the results of the first two studies with a different teacher and a differing student population. Because the controls used in this study involved regular classes in formal mathematics and programming, it also investigated the differing efficacies of that instruction vs. regular instruction in domains typically prescribed for the teaching and learning of problem solving. Study Three, then, explored the following questions:

1. Does explicit instruction and mediated Logo programming practice support the teaching and learning of five particular problem solving strategies -- subgoals formation, forward chaining, systematic trial and error, alternative representation, and analogy -- among high school students?

2. Is explicit problem solving instruction with Logo programming practice superior to regular instruction in mathematics and programming for supporting the development of such strategies?

Subjects.

Subjects were 40 eleventh and twelfth grade students at an American school in Switzerland enrolled in one of three classes -- a Logo class, an Advanced Placement (AP) Pascal class, or a Pre-Calculus class. No subjects had any previous Logo programming experience.

Methodology.

All subjects were given paper-and-pencil tests of their ability to apply the five problem solving strategies on which subjects improved in the first study. Subjects in the Logo class received explicit problem solving instruction and Logo programming practice in each strategy. Subjects in the AP Pascal and Pre-Calculus classes received regular content area instruction. Mean pre-test scores were compared between groups using one-way analysis of variance and found not to be statistically equivalent ($F_{2,37} = 12.76$, $p < .01$), hence groups could not be assumed generally equal in problem solving ability before treatment. An examination of group means revealed that the Logo group scored significantly lower than students in both the AP Pascal and

Pre-Calculus groups on pre-test measures. On completion of all treatments, subjects were post-tested using different but analogous tests. Raw scores on all tests except those for alternative representation were converted to percent correct scores and compared using a three-way analysis of variance. Independent variables were test, strategy, and group. The dependent variables were scores on the strategies tests. Because they had no maximum possible correct, alternative representations measures were evaluated seperately using a two-way analysis of variance.

Results.

Significant differences in pre- to post-test increases were found between groups. Further analysis of this finding revealed that subjects in the Logo class showed significantly improved subgoals formation, forward chaining, and systematic trial and error strategies. Increased ability in applying alternative representation strategies was also indicated but not conclusively demonstrated for this group. The results argue for the superiority of explicit strategy training and Logo programming practice over regular instruction in subjects traditionally prescribed for the teaching and learning of problem solving and demonstrate the efficacy of the instruction we developed with a very different student population.

TABLE 5
Study Three; ANOVA Table for Subgoals Formation, Forward Chaining, Backward Chaining, Systematic Trial & Error, and Analogy

	SS	DF	MS	F	PROB
MEAN	1220349.2	1	1220349.2	1014.79	0.0000
GROUP	1509.4	2	754.7	0.63	0.5395
ERROR	44494.6	37	1202.6		
TEST	1540.2	1	1540.2	10.07	0.0030
TG	3536.4	2	1768.2	11.56	0.0001
ERROR	5657.4	37	152.9		
STRATEGY	29113.7	3	9704.6	29.56	0.0000
SG	8144.3	6	1357.4	4.13	0.0009
ERROR	36444.4	111	328.3		
TS	1483.3	3	494.4	3.54	0.0170
TSG	1083.8	6	180.5	1.29	0.2663
ERROR	15493.5	111	139.6		

TABLE 5 shows the results of the three-way analysis of variance for subgoals formation, forward chaining, systematic trial and error, and analogy in Study Three. Significant main effects were found for test ($F_{2,37} = 10.07$, $p < .01$) and strategy ($F_{3,111} = 29.56$, $p < .01$), but not for group ($F_{2,37} = 0.63$, $p > .10$). Significant test by group ($F_{2,37} = 11.56$, $p < .01$), test by strategy ($F_{3,111} = 3.54$, $p < .05$), and strategy by group ($F_{6,111} = 4.13$, $p < .01$) interactions were also found. The results of the analysis of variance of scores on alternative representation measures is shown in TABLE 6. It reveals significant main effects for group ($F_{1,37} = 7.69$, $p <$

.01) and test ($F_{1,37} = 5.86$, $p < .05$), but no test by group interaction ($F_{2,37} = 1.65$, $p < .10$).

TABLE 6
Study Three; ANOVA Table for Alternative Representation

	SS	DF	MS	F	PROB
MEAN	700315.1	1	700315.1	300.29	0.0000
GROUP	35877.8	2	17938.9	7.69	0.0016
ERROR	86290.1	37	2332.2		
TEST	4310.8	1	24819.1	5.86	0.0205
TG	2431.2	2	1215.6	1.65	0.2055
ERROR	27228.5	37	735.9		

The main test and strategy effects are not particularly meaningful as previously explained. Indeed, the group effect found for alternative representation measures is not meaningful either because the groups were not equivalent to begin with. The lack of a group effect in the three-way analysis of variance, however, is meaningful because it reveals that groups which were not statistically equivalent before treatment became equivalent after treatment. Because students in the Logo group scored lower on pre-test measures than students in the other two groups, the lack of a group effect indicates an improvement in their scores resulting from the intervention. This result is corroborated by the finding of a test by group interaction which indicates differences in pre- to post-test changes resulting from differing treatments. This interaction was examined in greater detail by assessing the simple tests effect at each level of group. A strong test effect was found for the Logo group ($F_{1,37} = 46.99$, $p < .01$), but no test effect was found for the other two groups, indicating that the Logo group, and the Logo group alone, improved across all four strategy measures tested.

***** INSERT FIGURE 6 ABOUT HERE *****

FIGURE 6 illustrates these differences. Notice that students in the Logo group appear to have significantly increased on all measures except analogy. Indeed, looking at the simple test effect at each level of group and strategy, we discovered a strong test effect for students in the Logo group on subgoals formation, forward chaining, and systematic trial and error measures ($p < .01$), but no test effect at all for analogy measures ($p > .10$). A possible explanation for the lack of improvement on tests of analogy (especially considering that older students in the first study improved the most on these measures) is that the analogy test itself was too easy for the high school students tested in Study Three. Indeed, students in this study scored so high on the analogy pre-test that there was little room for improvement on the post-test.

A significant simple test effect was also found for students in the Pre-Calculus group on the systematic trial and error measure ($p < .05$), indicating that students in this group developed systematic trial and error strategies to some extent, but significant differences were still found between this group and the Logo group on the systematic trial and error measure. No simple test

effects were found on any other strategy measures for members of the Pre-Calculus group, or on any strategy measures for members of the AP Pascal group. We can conclude, then, that the intervention we designed was effective in increasing participating students' subgoals formation, forward chaining, and systematic trial and error problem solving skills, and that it was more effective in this respect than regular instruction in subject areas traditionally prescribed for the teaching and learning of problem solving. An examination of the simple test effect at each level of group for alternative representation measures also reveals a strong test effect for the Logo group ($p < .01$), but not for the other two groups ($p > .10$). Because no test by group interaction was found in the analysis of variance for this measure, however, the most we can conclude is that it is possible that students in the Logo group showed greater increases on alternative representation measures as a result of the interventio we designed.

Discussion

The results of Study Three support the efficacy of the instruction we designed for the teaching and learning of subgoals formation, forward chaining, and systematic trial and error strategies among high school students, and argue for its superiority over regular instruction in subjects typically prescribed for such teaching and learning. Indications are that explicit instruction and Logo programming practice is also effective for the development of alternative representation strategies among such population. The results thus strongly support Thorndike's (1907) contention that the transfer of problem solving skills from formal disciplines does not occur automatically, but rather occurs to the extent that such skills are explicitly taught and practiced within such formal study.

CONCLUSIONS

The results of our research demonstrate the effectiveness of a knowledge-based approach for developing four problem solving strategies -- subgoals formation, forward chaining, systematic trial and error, and analogy -- through Logo programming practice. Indications are that alternative representation strategies may also be similarly developed. They demonstrate that explicit instruction and Logo programming practice is more effective than explicit instruction with concrete manipulatives practice, Logo discovery learning, and instruction in subjects traditionally prescribed for the teaching and learning of problem solving. They thus suggest that Papert (1980) was correct in claiming that computing environments can uniquely support the development of problem solving and critical thinking skills, but that he was incorrect in assuming that such development would take place without formal instruction.

That explicit instruction is integral to the development of problem solving skills through Logo programming is shown by Studies Two and Three. These findings are corroborated by other research (Carver, 1987; Lehrer & Randle, 1987; Thompson & Wang, 1988; de Corte et al, 1989;

Lehrer et al, 1989; Swan & Black, 1989). We believe the reason why such instruction is necessary is that students need generalizable declarative knowledge of particular problem solving strategies in order to decontextualize such knowledge from programming and apply it in other domains. In the instruction we designed, problem solving strategies were broken into their component steps and explicitly taught. Students were thus provided with declarative knowledge of the problem solving strategies to be learned.

Declarative knowledge of particular problem solving strategies, however, is not in itself enough to ensure their development, as shown by Study Two. Procedural knowledge is also necessary. It is our belief that Seymour Papert (1980) is right; that computing environments, Logo programming environments in particular, are uniquely conducive to the development of such skills because they support quasi-concrete representations of these abstract strategies that students can inspect, manipulate, and test through practice. In this vein, it is instructive that backward chaining, alternative representation, and analogy strategies, for which their are no direct Logo representations, were the least likely to be developed by the students in our studies, while subgoals formation, forward chaining and systematic trial and error strategies, which are the most concretely represented in the language, were the most likely to be developed. In particular, no instructional effects at all were found for backward chaining strategies, most likely because it is not possible to develop programs backwards, hence, students had no procedural analog to link to the declarative knowledge with which they were provided. Likewise, the cut-paper manipulatives in Study Two probably did not provide students in that condition with anything like the quasi-concrete strategy models available in Logo programming.

The development of problem solving and critical thinking skills is a crucial problem for education today. The research presented in this paper offers an instructional intervention for developing particular problem solving abilities. More importantly, it suggests that the Logo programming environment in particular, computing environments in general, can perform a mediating role in the development of problem solving abilities by supporting quasi-concrete, dynamic representations of abstract ideas which can help students bridge the gap between concrete and formal thought when students are supplied with a declarative understanding of particular problem solving skills through explicit knowledge-based instruction. If computing environments can indeed be designed to support such transitional objects for thinking, they might play an important role in the teaching and learning of problem solving. In today's educational climate, the notion certainly deserves further investigation.

REFERENCES

Anderson, J. R. (1983) The Architecture of Cognition. Cambridge, MA: Harvard University Press.

Black, J. B., Swan, K. & Schwartz, D. (1988) Developing thinking skills with computers. Teachers College Record, 89 (3).

Bolter, J. D. (1984) Turing's Man. Chapel Hill, NC: University of North Carolina Press.

Carver, S. M. (1987) Transfer of Logo debugging skill: analysis, instruction, and assessment. Computer Systems Group Bulletin, 14 (1), 4-6.

Carver, S. M. & Klahr, D. (1986) Assessing children's Logo debugging skills with a formal model. Journal of Educational Computing Research, 2 (4, 487-525.

Clement, C. A., Kurland, D. M., Mawby, R. & Pea, R. D. (1986) Analogical reasoning and computer programming. Journal of Educational Computing Research, 2 (4), 423-454.

Clements, D. H. (1987) Longitudinal study of the effects of Logo programming on cognitive abilities and achievement. Journal of Educational Computing Research, 3 (1), 73-94.

Clements, D. H. & Gullo, D. F. (1984) Effects of computer programming on young children's cognition. Journal of Educational Psychology, 76, 1051-1058.

Corno, L (1986) The metacognitive control components of self-regulated learning. Contemporary Educational Psychology, 11, 333-346.

De Corte, E., Verschaffel, L. & Schrooten, H. (1989) Cognitive Effects of Learning to Program in Logo: A One-Year Study with Sixth Graders. Leuven, Belgium: Project: Computers and Thinking, Center for Instructional Psychology, Katholieke Universiteit Leuven.

Delclos, V. R., Littlefield, J. & Bransford, J. D. (1985) Teaching thinking through Logo: the importance of method. Roeper Review, 7 (3), 153-156.

Degelman, D., Free, J. U., Scarlato, M., Blackburn, J. M. & Golden, T. (1986) Journal of Educational Computing Research, 2 (2), 199-205.

Duncker, K. (1945) On problem solving. Psychological Monograph, 58, (6).

Ehrlich, K. Abbott, V., Salter, W. & Soloway, E. (1984) Issues and problems in studying transfer effects of programming. Paper presented at the annual meeting of the American Educational Research Association. New Orleans, April, 1984.

Feuerstein, R. (1980) InstrumentalEnrichment. Baltimore, MD: University Park Press.

Feurzig, W., Horowitz, A. & Nickerson, R. S. (1981) MicroComputers in Education (Report No. 4798). Cambridge, MA: Bolt, Baranek & Newmann.

Gentner, D. (1987) Mechanisms of Analogical Learning. Urbana, IL: Department of Computer Science, University of Illinois.

Ginsburg, H. & Opper, S. (1980) Piaget's Theory of Intellectual Development. Englewood Cliffs, NJ: Prentice-Hall.

BEST COPY AVAILABLE

Gorman, H., Jr. & Bourne, L. E. (1983) Learning to think by learning Logo: rule learning in third-grade computer programmers. Bulletin of the Psychonomic Society, 21, 165-167.

Greeno, J. G. and Simon, H. A. (1984) Problem Solving and Reasoning (Technical Report No. UPITT/LRDC/ONR/APS-14). Washington, DC: Learning Research and Development Center, Office of Naval Research.

Harvey, B. (1982) Why Logo? Byte, 7 (8), 92-95.

Holyoak, K. J. & Koh, K. (1987) Surface and structural similarity in analogical transfer. Memory and Cognition, 15, 332-340.

Lawler, R. W. (1985) Computer Experience and Cognitive Development: A Child's Learning in a Computer Culture. New York: Halsted Press.

Lehrer, R. and Randle, L. (1987) Problem solving, metacognition and composition: the effects of interactive software for first-grade children. Journal of Educational Computing Research, 3 (4), 409-428.

Lehrer, R., Sancilio, L. and Randle, L. (1989) Learning pre-proof geometry with Logo. Cognition and Instruction, 6 (2), 159-184.

Leron, U. (1985) Logo today: vision and reality. The Computing Teacher, 12 (6), 26-32.

Mandinach, E. B. & Linn, M. C. (1987) Cognitive consequences of programming: achievements of experienced and talented programmers. Journal of Educational Computing Research, 3 (1), 53-72.

Mayer, R. E., Dyck, J. L. & Vilberg, W. (1986) Learning to program and learning to think: what's the connection? Communications of the ACM, 29 (7), 605-610.

McLuhan, M. (1964) UnderstandingMedia. NY: New American Library.

Miller, G. E. & Emihovich, C. (1986) The effects of mediated programming instruction on preschool children's self-monitoring. Journal of Educational Computing Research, 2 (3), 283-297.

Newell, A. & Simon, H. A. (1972) Human Problem Solving. Englewood Cliffs, NJ: Prentice-Hall.

Papert, S. (1980) Mindstorms. New York: Basic Books.

Papert, S., Watt, D., diSessa, A. & Weir, S. (1979) Final Report of the Brookline Logo Project (Logo Memo 53). Cambridge, MA: MIT Artificial Intelligence Laboratory.

Pea, R. D. & Kurland, D. M. (1984) On the cognitive effects of learning computer programming. New Ideas in Psychology, 2 (2), 137-167.

Polya, G. (1973) How To Solve It. Princeton, NJ: Princeton University Press.

Salomon, G. (1981) Interaction of Media, Cognition, and Learning. San Francisco: Jossey-Bass.

Salomon, G. & Perkins, D. N. (1987) Transfer of cognitive skills from programming: When and how? Journal of Educational Computing Research, 3 (2), 149-170.

Shaw, D. G. (1986) Effects of learning to program a computer in BASIC or Logo on problem solving abilities. AEDS Journal, 19 (2/3) 176-189.

Soloway, E. (1986) Why Kids Should Learn To Program. New Haven, CT: Cognition and Programming Project, Department of Computer Science, Yale University.

Statz, J. (1973) Problem Solving and Logo: Final Report of the Syracuse Logo Project. Syracuse, NY: Syracuse University.

Swan, K. & Black, J. B. (1989) Logo Programming and the Teaching and Learning of Problem Solving (CCTE Report 89-1). New York: Teachers College, Columbia University.

Thompson, A. D. & Wang, H. M. C. (1988) Effects of a Logo microworld on student ability to transfer a concept. Journal of Educational Computing Research, 4 (3), 335-347.

Thorndike, E. L. (1907) Elements of Psychology. New York: Teachers College Press.

Watt, D. (1982) Logo in the schools. Byte, 7 (8), 116-134.

Wicklegren, W. A. (1974) How to Solve Problems. San Francisco: W. H. Freeman.

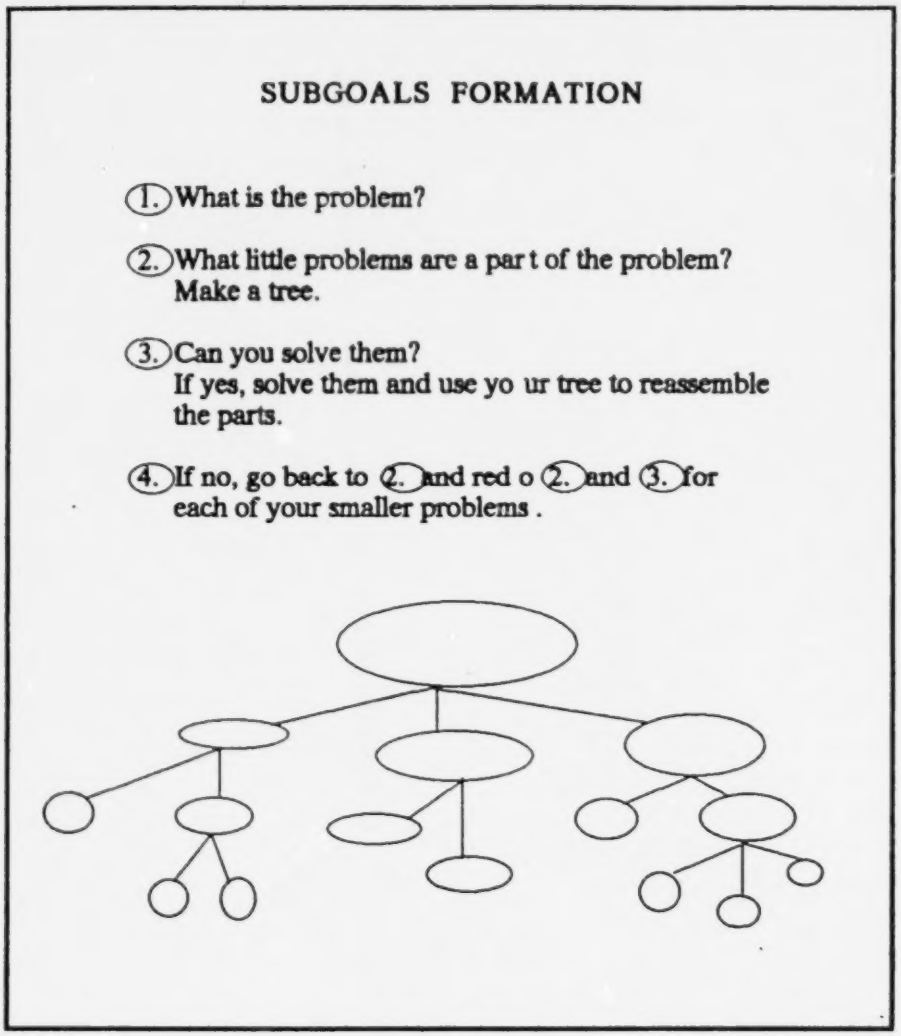
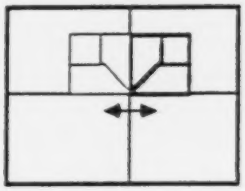

SUBGOALS FORMATION
1. What is the problem?
2. What little problems are a par t of the problem?
Make a tree.
3. Can you solve them?
If yes, solve them and use yo ur tree to reassemble
the parts.
4. If no, go back to 2. and red o 2. and 3. for
each of your smaller problems .

FIGURE 1
Wall Chart for Subgoals Formation

analogy

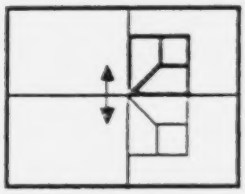
1. Horizontal symmetry

Put together at least 3 shapes to create a design in the upper right quadrant of the screen, then draw the mirror image of your design in the lower right quadrant.



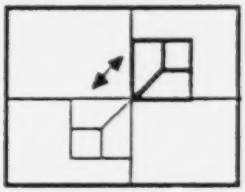
2. Vertical symmetry

Put together at least 3 shapes to create a design in the upper right quadrant of the screen, then draw the mirror image of your design in the lower left quadrant.



3. Diagonal symmetry

Put together at least 3 shapes to create a design in the upper right quadrant of the screen, then draw the mirror image of your design in the lower right quadrant.



4. Mirrors

Put together at least 3 shapes to create a design in the upper right quadrant of the screen, then draw the mirror images of your design in the other three quadrants.

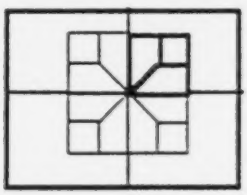
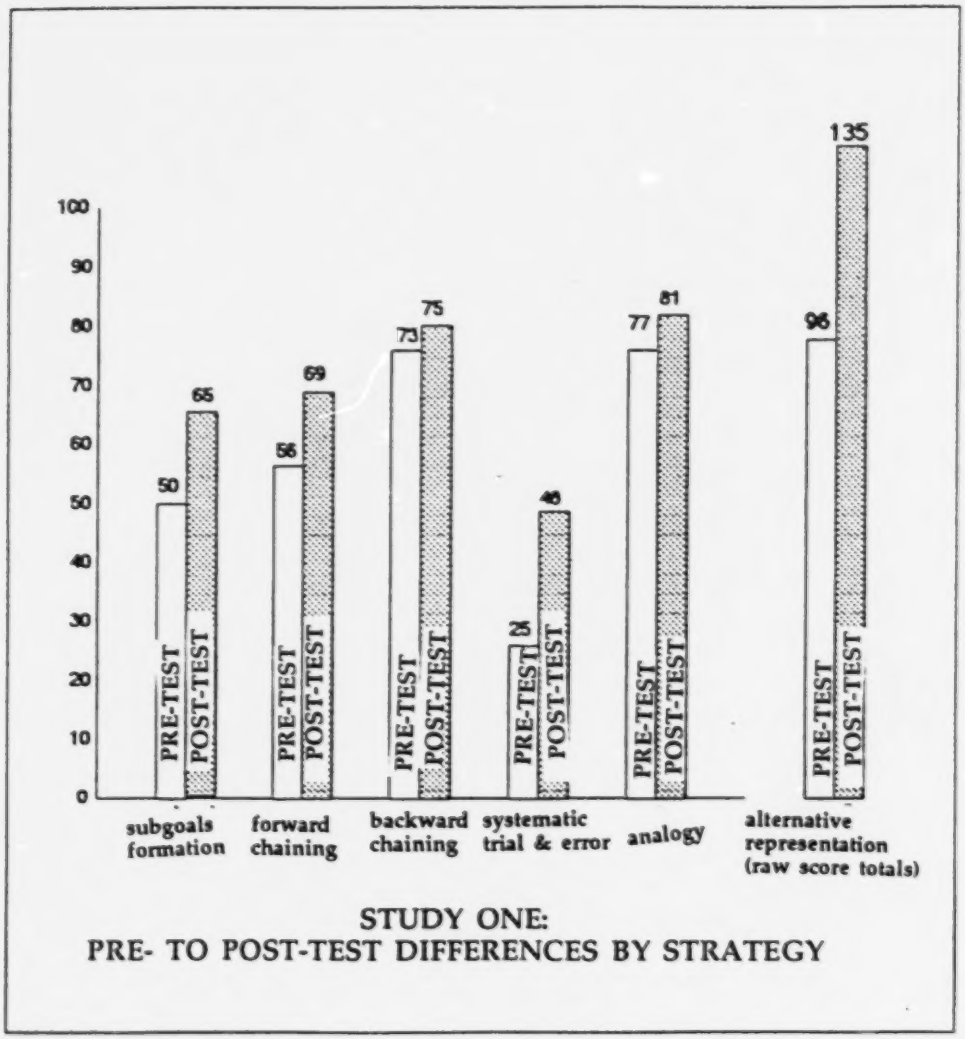
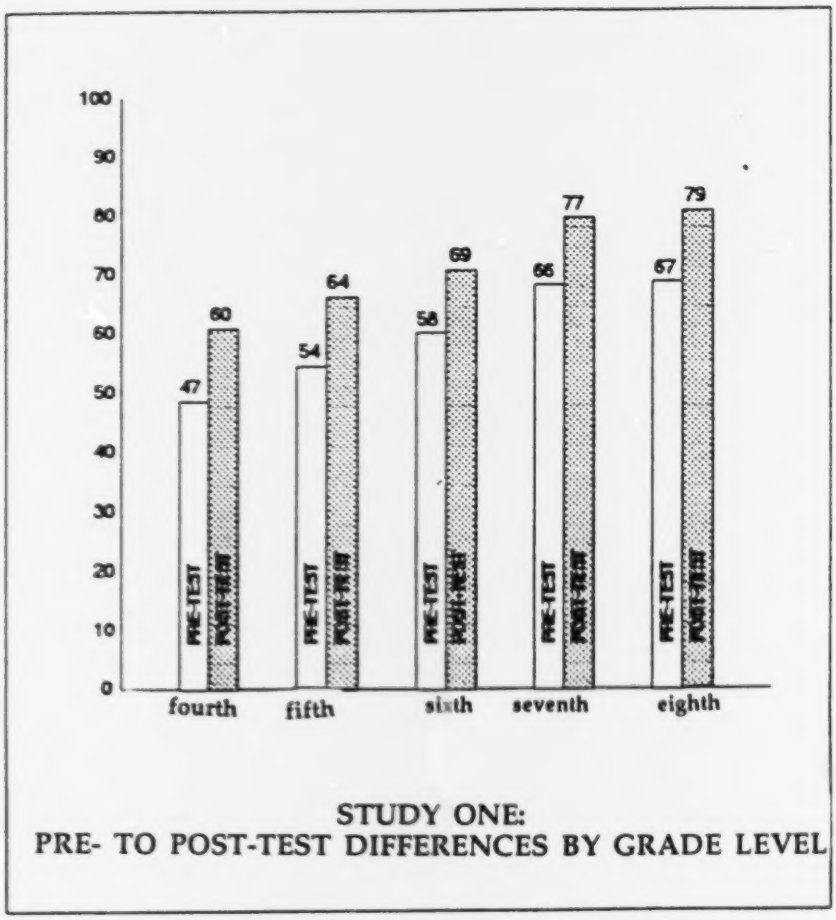


FIGURE 2
Problem Set for Analogy



STUDY ONE:
PRE- TO POST-TEST DIFFERENCES BY STRATEGY

FIGURE 3



STUDY ONE:
PRE- TO POST-TEST DIFFERENCES BY GRADE LEVEL

FIGURE 4

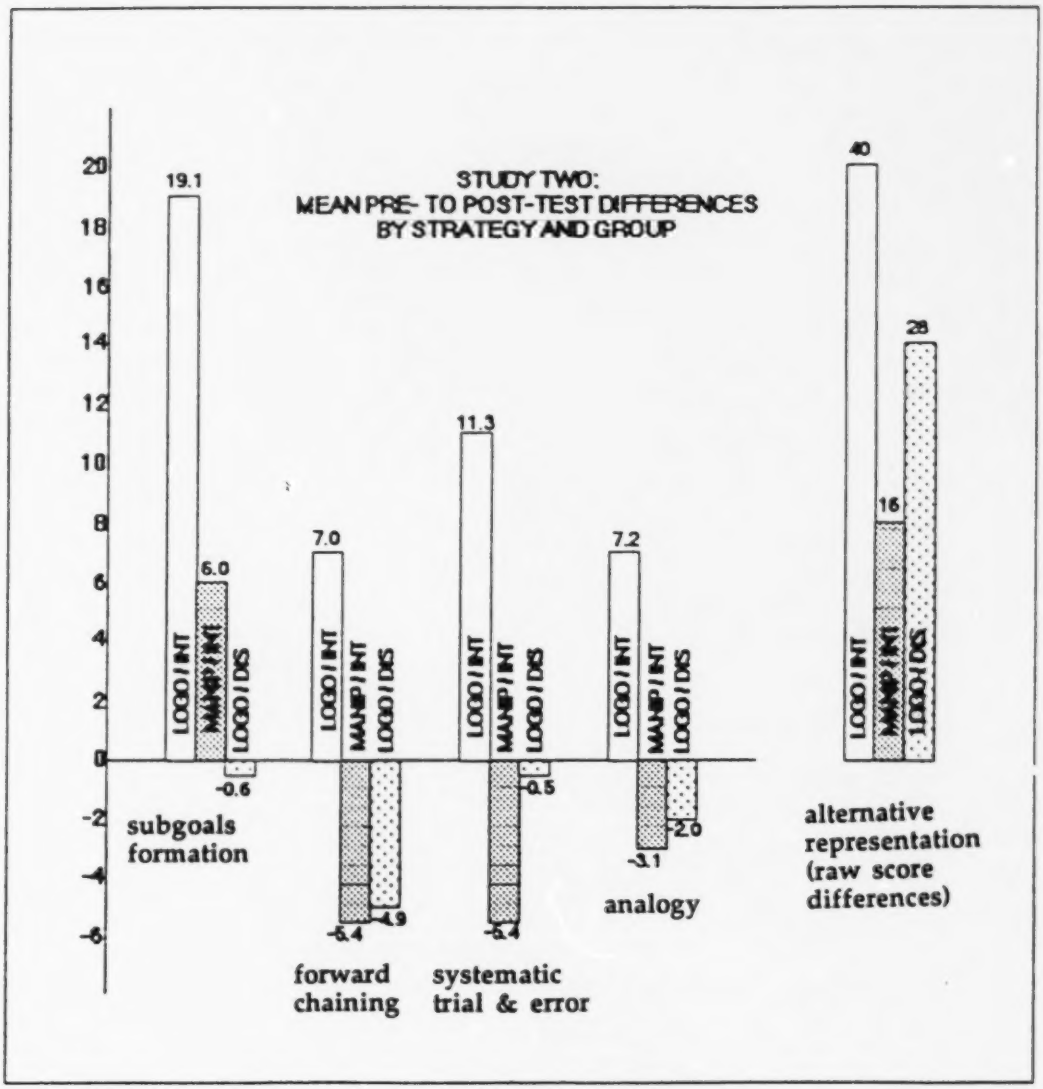

STUDY TWO:
MEAN PRE- TO POST-TEST DIFFERENCES
BY STRATEGY AND GROUP
20
18
16
14
12
10
8
6
4
2
0
-2
-4
-6
19.1
6.0
-0.6
7.0
-5.4
-4.9
11.3
-5.4
-0.5
7.2
-3.1
-2.0
40
16
28
LOGO I INT
MANIP INT
LOGO I DIS
subgoals formation
forward chaining
systematic trial & error
analogy
alternative representation (raw score differences)

FIGURE 5

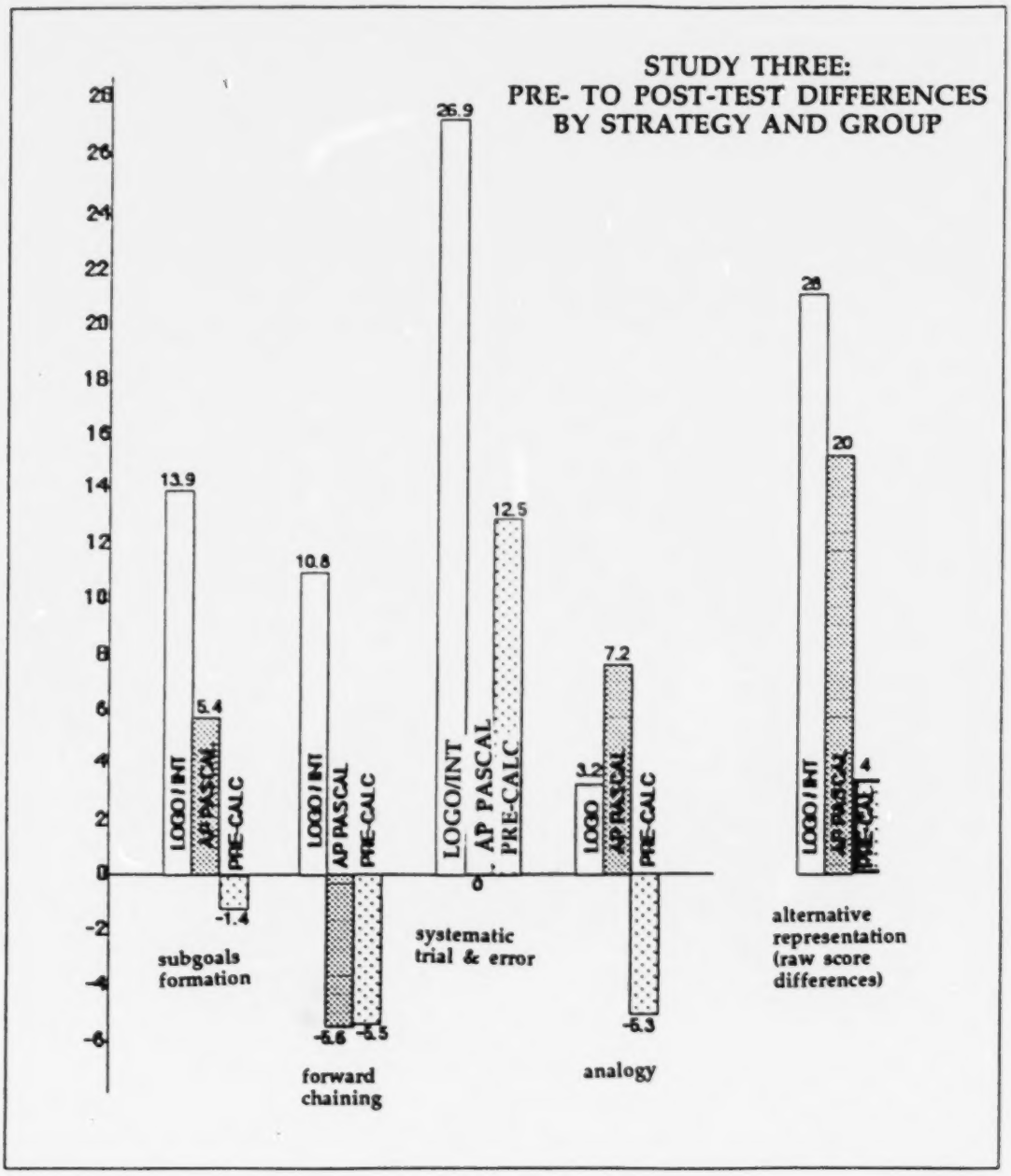

STUDY THREE:
PRE- TO POST-TEST DIFFERENCES
BY STRATEGY AND GROUP
LOGO/INT
AP PASCAL
PRE-CALC
13.9
5.4
-1.4
10.8
-6.6
-6.5
26.9
12.5
0
7.2
-6.3
28
20
4
subgoals formation
forward chaining
systematic trial & error
analogy
alternative representation (raw score differences)

FIGURE 6

APPENDIX

Problem Solving Strategy Measures

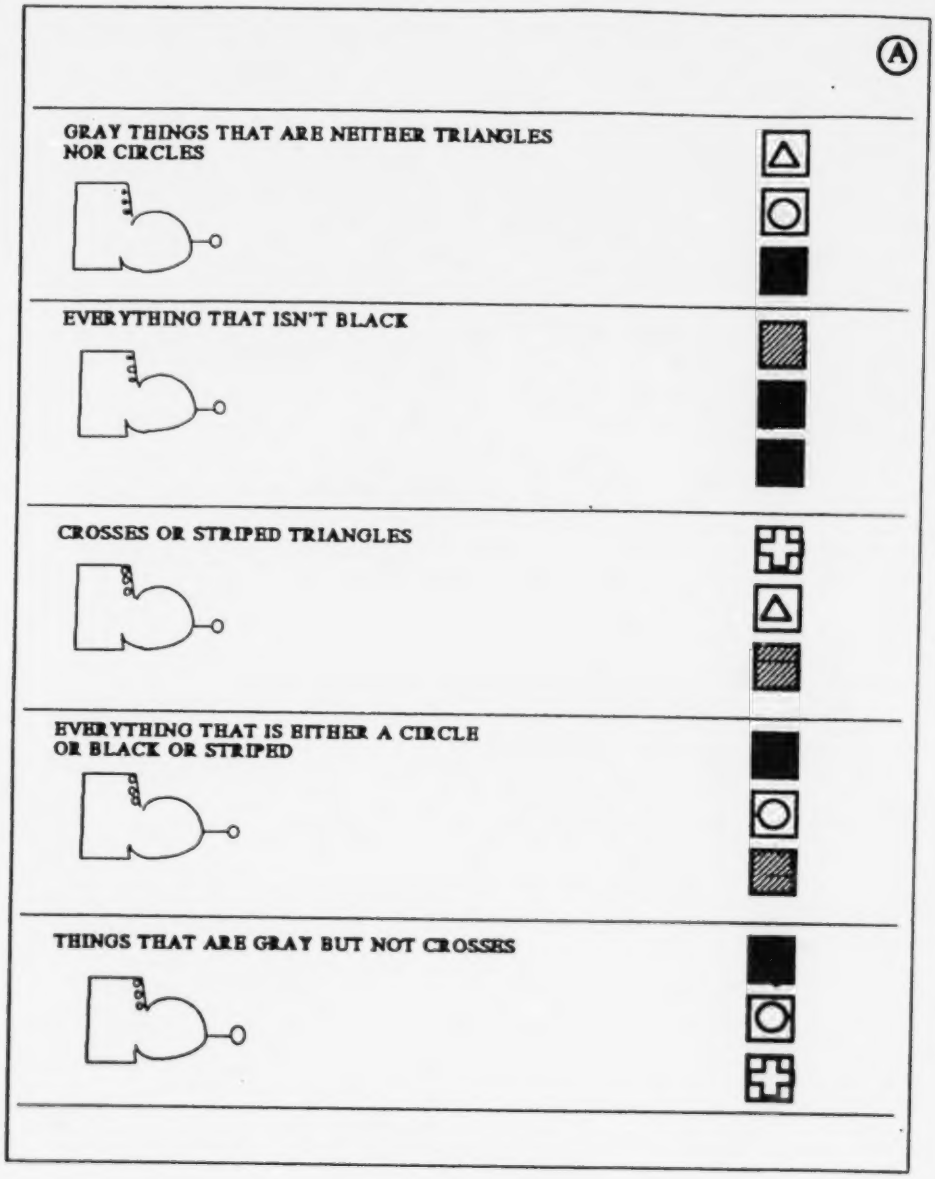
(A)

3. The Sport Shop sold sneakers for one week at $30 a pair and made $1800.00. The next week they reduced the price to $20.00 a pair and sold twice as many. How much money did they make altogether?

4. Jean and Marie split a milk shake that cost $1.00. Marie didn't have quite enough money to pay for her half. She still owes Jean $.10. How much money did Jean put in for the milk shake?

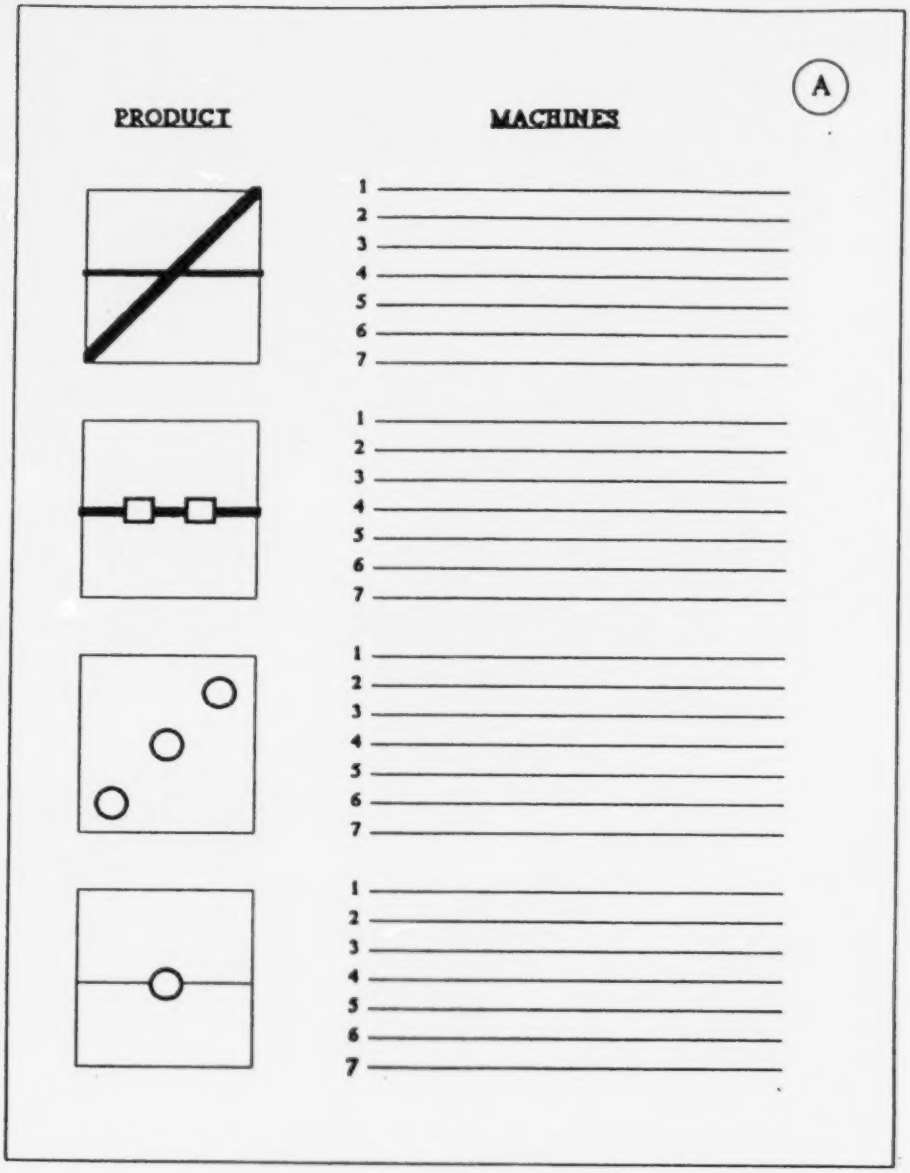
Subgoals formation.

Our measure of students' ability to decompose complex problems into smaller subgoals units consisted of five mathematical word problems that required decomposition for correct solution. S dents were asked not only to solve the problems but to show how they broke them into parts. They were given credit for correctly identified subgoals as well as for the correct answer, with a possible total of five points per question.


A
GRAY THINGS THAT ARE NEITHER TRIANGLES NOR CIRCLES
EVERYTHING THAT ISN'T BLACK
CROSSES OR STRIPED TRIANGLES
EVERYTHING THAT IS EITHER A CIRCLE OR BLACK OR STRIPED
THINGS THAT ARE GRAY BUT NOT CROSSES

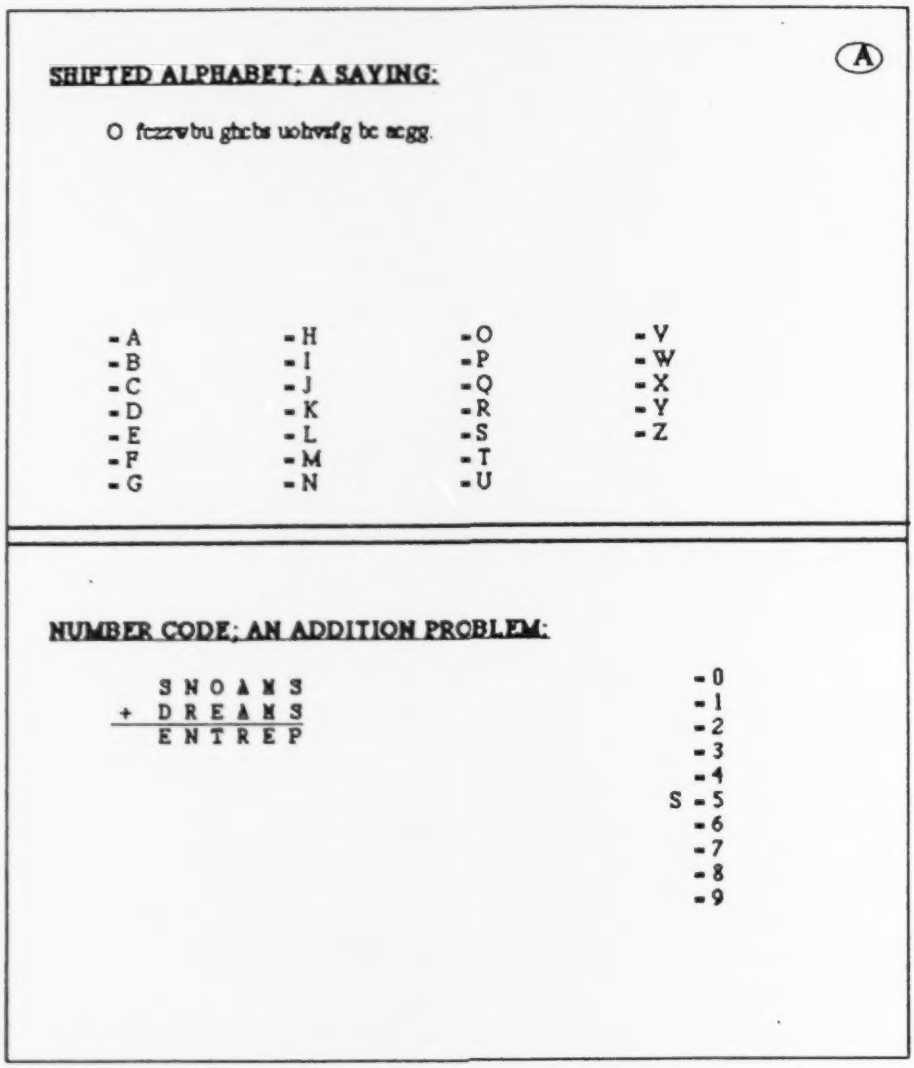
Forward Chaining.

The test designed to measure subjects' forward chaining skills was a paper-and-pencil version of the computer game, Rocky's Books (The Learning Company, 1982). In Rocky's Boots, symbolic "AND", "OR", and "NOT" gates are combined to produce machines that respond to targeted attributes and sets of attributes (e.g., gray triangles, corsses or striped circles, everything that is not black, etc.). Combinations of gates must be built up in a forward chaining manner to achieve correct solutions. Our paper-and-pencil version had subjects draw the required connections. There was a total of fifteen questions which were given one point each for correct solution.



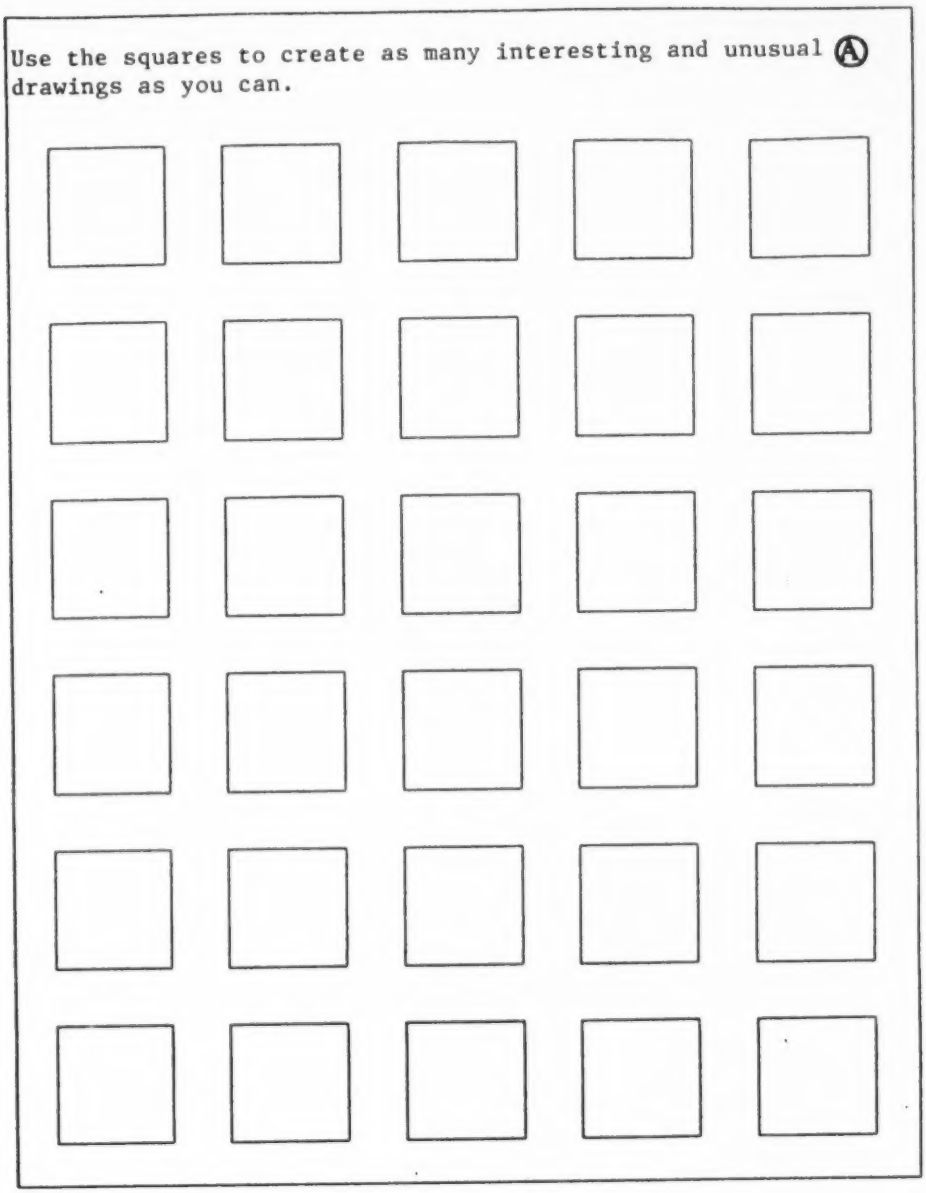
Backward chaining.

The test designed to measure subjects' backward chaining skills was a paper-and-pencil adaptation of the computer game, The Factory (Sunburst, 1984). In The Factory, players are shown a finished product and asked to combine various machines to produce a similar product. Thus, players must work backwards from the product to deduce a correct sequence of machines that will produce it. Our paper-and-pencil version had subjects list the required machine sequences. The test consisted of fifteen questions which were given one point each for correct solution.



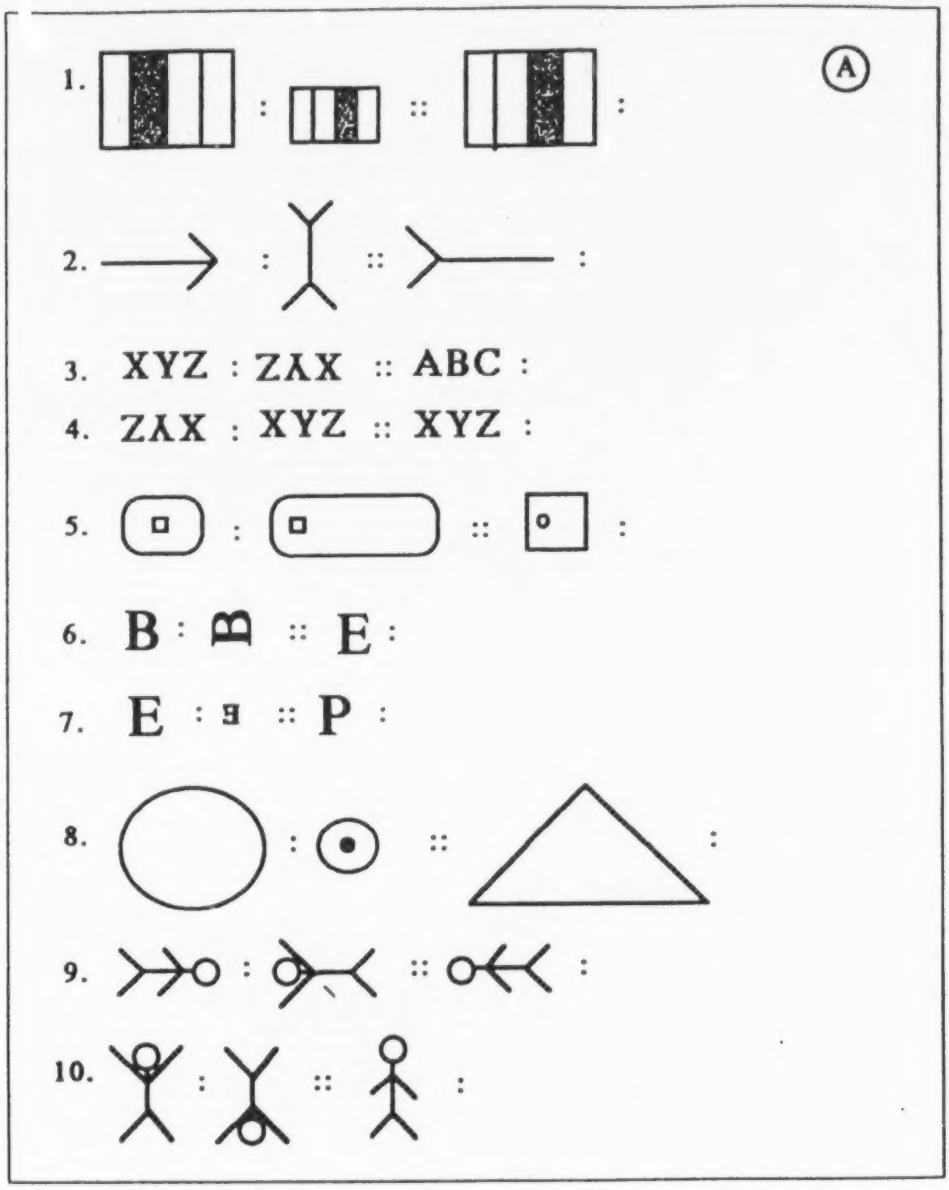
Systematic trial and error.

Cryptography involved systematically trying and testing different symbol combinations to attain coherent decoding systems. We chose two decoding exercises to test subjects' abilities to systematically utilize trial and error strategies. The first of these was a shifted alphabetical code. The second involved variations on a number code problem devised by Newell and Simon (1971). Students were given ten points for correctly decoded problem. For partial solutions on the shifted alphabet problem, one half point was given for each correctly identified letter. On the number code problem, a full point was given for each correctly identified letter.



Alternative representation

The measure of subjects' ability to create alternative representation used was derived from the Torrance Test of Creative Thinking (Torrance, 1972). Students were given sets of geometric fitures (squares or circles) and asked to use these to produce as many interesting and unusual drawings as they could. The resultant drawings were scored for quantitiy, diversity, originality, and elaboration.



Analogy

Subjects' skill at analogical reasoning was measured with completion exercises consisting of ten verbal and ten visual analogies. Students were given one point for each correct answer.